# A1

A1_2.sh - Write a shell script which takes as parameters two names of text files. The script will compare the two text files line by line and display the first 3 text lines that differ.

```
#!/bin/bash
# Check if correct number of arguments provided
if [ "$#" -ne 2 ]; then
    echo "not 2 args"
    exit 1
fi
file1="$1"
file2="$2"
# Check if files exist
if [ ! -e "$file1" ] || [ ! -e "$file2" ]; then
    echo "Error: One or both files do not exist."
    exit 1
Fi
#Check if files are empty
if [ ! -s "$file1" ]; then
            echo "Error: File '$file1' is empty."
           exit 1
fi
if [ ! -s "$file2" ]; then
            echo"Error: File '$file2' is empty."
            exit 1
fi
# Compare files line by line
line_num=1
count=0
while IFS= read -r line1 && IFS= read -r line2 <&3; do
    if [ "$line1" != "$line2" ]; then
        echo "Line $line_num:"
        echo "$file1: $line1"
        echo "$file2: $line2"
        count=$((count+1))
        if [ "$count" -eq 3 ]; then
            break
        fi
    fi
    line_num=$((line_num+1))
done < "$file1" 3< "$file2"
# Display message based on number of differences found
if [ "$count" -eq 0 ]; then
    echo "No differences found."
elif [ "$count" -eq 1 ]; then
    echo "Only one difference found."
```

```
elif [ "$count" -eq 2 ]; then
    echo "Only two differences found."
fi
```

## A1_4_v2.sh - Write a shell script that reads numbers from the keyboard (the reading stops when the user has entered the number 0). The script will display the sum of numbers entered from the keyboard.

```
#!/bin/bash
#!/bin/bash

echo "This is a script that will compute the sum of all numbers given from the keyboard until 0 is entered. The program stops and shows the result."

sum=0
i=1

while true; do
    read -p "Enter number $i (0 to stop): " number

    if [[ -z "$number" ]]; then
        echo "Error: Number cannot be empty"
        continue
    fi

    if [[ "$number" =~ ^[0-9]+$ ]]; then   # ^-beginning   $-end   +-2 or more digits/characters
        if [[ "$number" -eq 0 ]]; then
            break
        fi
        sum=$((sum + number))
        ((i++))
    else
        echo "Error: Invalid number input"
        continue
    fi
done

echo "The result is: $sum"
```

## A1_4.sh

```
#!/bin/sh
echo "Enter the numbers"
suma=0
while true;
do
    read x
    if [ "$x" -eq 0 ];
    then
```

```
        break
    fi
    suma=$((suma+x))
done
echo "Suma: $suma"
```

## A1_8.sh - Write a shell script which takes as parameters two file names (a file that contains usernames and a file that contains any text). The script will send a mail to each user in the first file (the mail message will be the text in the second file).

```
#!/bin/bash

# Check if exactly two arguments are provided
if [ "$#" -ne 2 ]; then
    echo "Usage: $0 <usernames_file> <message_file>"
    exit 1
fi

# Check if the first argument is a file
if [ ! -f "$1" ]; then
    echo "Error: File '$1' does not exist."
    exit 1
fi

# Check if the second argument is a file
if [ ! -f "$2" ]; then
    echo "Error: File '$2' does not exist."
    exit 1
fi

usernames_file="$1"
message_file="$2"

# Read the message content once
message=$(<"$message_file")

# Send email to each user listed in the usernames file
while IFS= read -r username; do     #read the content of usernames_file in the variable username
    if [ -n "$username" ]; then
        echo "$message" | mail -s "Mail sent through command line" "$username"
        echo "Mail sent to $username"
    fi
done < "$usernames_file"

exit 0
```

## A1_11.sh - mine - Write a shell script which sorts files

```
echo "List by their names: "
```

```
ls -l | sort -k 9  #ninth field, which is the file name.k-to specify
echo -e "\nListing files sorted by last modified date: "
ls -lt
echo -e "\nListing files sorted by their file size: "
ls -lS
```

## A1_12_v2.sh - Write a shell script which takes as parameter a directory name. The script will display the filename and the first 3 lines of each ASCII text file found in that directory

```
if [ $# -ne 1 ]; then
    echo "Usage: $0 directory_name"
    exit 1
fi
if [ ! -d "$1" ]; then
    echo "$1 is not a directory"
    exit 1
fi
for file in "$1"/*;  #Loops through all files in the specified directory.
do              #-b: Suppresses the filename in the output.
    if [ -f "$file" ] && [ "$(file -b --mime-type "$file")" = "text/plain" ]; then
        echo "Filename: $file"
        head -n 3 "$file"
        echo
    fi
done
```

## A1_14.sh - Write a shell script which displays all files in the current directory and its subdirectories that have write permission for the group of which the owner belongs.

```
check_permission() {
    file="$1"
    if [ -w "$file" ]; then
        echo "$file"
    fi
}
traverse_dir() {
    directory="$1"
    for item in "$directory"/*; do
        if [ -d "$item" ]; then     #d -if directory        f -if file
            traverse_dir "$item"
        elif [ -f "$item" ]; then
            check_permission "$item"
        fi
    done
}
traverse_dir "$(pwd)"
```

# A2

A2_5s.sh - Write a shell script which takes as parameters several file names. The script will delete all words that contain at least one digit from all given files.

```
#!/bin/bash
for file in "$@"; do
    if [ -f "$file" ]; then
        # Use sed to delete words containing at least one digit
        sed -i -E 's/\b[[:alpha:]]*[[:digit:]][[:alnum:]]*\b//g' "$file"
        echo "Words containing digits deleted from $file"
    else
        echo "File $file not found."
    fi
done
```

A2_6s_v2.sh - Write a shell script which takes as parameters a lowercase letter followed by several file names. The script will replace any special character with the given letter in all files given as parameters.

```
#!/bin/bash
# Check if the number of arguments is less than 2
if [ "$#" -lt 2 ]; then
    echo "Usage: $0 <lowercase_letter> <file1> [<file2> ...]"
    exit 1
fi
# Extract the lowercase letter from the first argument
letter="$1"
# Check if the first argument is a lowercase letter
if ! [[ "$letter" =~ ^[a-z]$ ]]; then
    echo "Error: First argument must be a lowercase letter."
    exit 1
fi
# Check if the lowercase letter is a single character
if [ ${#letter} -ne 1 ]; then
    echo "Error: First argument must be a single lowercase letter."
    exit 1
fi
# Check if the remaining arguments are files
for file in "${@:2}"; do
    if [ ! -f "$file" ]; then
        echo "Error: '$file' is not a file."
        exit 1
    fi
done
# Shift to skip the first argument (the letter)
shift
# Loop through each file provided as argument
```

```
for file in "$@"; do
    # Check if the file exists
    if [ ! -f "$file" ]; then
        echo "File '$file' does not exist. Skipping..."
        continue
    fi
    # Perform the replacement - g -replace all occurrences of the pattern in each line, not just the first
sed -i "s/[^a-zA-Z0-9]/$letter/g" "$file"
    echo "Replaced special characters with '$letter' in file '$file'."
done
```

A2_9s.sh - Write a shell script which takes as parameters several file names. The script will delete the 2nd and 4th word in each line of the given files. The words shall contain only letters or numbers and shall be separated by spaces.

```
if [ "$#" -eq 0 ]; then
    echo "Please input file names when using the script."
    exit 1
fi
for file in "$@"; do
    if [ ! -f "$file" ]; then
        echo "File '$file' does not exist."
        continue
    fi
    #                1                          2                          3                          4
    sed -i -E
's/^(([a-zA-Z0-9]+[[:space:]]+){1})[a-zA-Z0-9]+[[:space:]]+(([a-zA-Z0-9]+[[:space:]]+){1})[a-zA-Z0-9
]+/\1\3/' "$file"
#\1\3: Replaces the matched part with the first and third captured groups,
    echo "Words removed successfully in '$file'"
Done
```

—---V2—----

```
    while IFS= read -r line; do
        # Split the line into words using an array
        words=($line) # 0 1 2 3 4

        # Remove the 2nd and 4th words if they exist
        if [ "${#words[@]}" -ge 4 ]; then
            words=("${words[0]}" "${words[2]}" "${words[@]:4}")
        elif [ "${#words[@]}" -ge 2 ]; then
            words=("${words[0]}" "${words[@]:2}")
        fi

        # Print the modified line
        echo "${words[*]}"
    done < "$file" > "${file}.tmp"

    # Replace the original file with the modified content
```

```
    mv "${file}.tmp" "$file"
done
```

A2_10s.sh - Write a shell script which takes as parameters several file names. The script will interchange the 1st word with 3rd word in each line of the given files. The words shall contain only letters or numbers and shall be separated by any other character.

```
# Check if at least one file name is provided
if [ "$#" -lt 1 ]; then
    echo "Usage: $0 <file1> [file2] ... [fileN]"
    exit 1
fi

# Iterate over each provided file name
for file in "$@"; do
    # Check if the file exists and is not empty
    if [ ! -s "$file" ]; then
        echo "Error: File '$file' does not exist or is empty."
        continue
    fi

    # Process each line of the file
    while IFS= read -r line; do
        # Use grep and sed to split the line into words and separators
        words=($(echo "$line" | grep -o -E '[a-zA-Z0-9]+'))
        separators=($(echo "$line" | grep -o -E '[^a-zA-Z0-9]+'))

        # Ensure we have enough words to swap
        if [ "${#words[@]}" -ge 3 ]; then
            # Swap the 1st and 3rd words
            temp="${words[0]}"
            words[0]="${words[2]}"
            words[2]="$temp"
        fi

        # Reconstruct the line
        new_line=""
        for (( i=0; i<${#words[@]}; i++ )); do
            new_line+="${words[i]}"
            if [ $i -lt ${#separators[@]} ]; then    #less than -to put in between
                new_line+="${separators[i]}"
            fi
        done

        # Print the modified line
        echo "$new_line"
    done < "$file" > "${file}.tmp"

    # Replace the original file with the modified content
    mv "${file}.tmp" "$file"
done
```

A2_12s.sh - Write a shell script which takes as parameters a lowercase letter followed by several file names. The script will replace each digit with the letter given as a parameter in all given files.

```
# I check if there are any arguments
if [[ "$#" == 0 ]]; then
	echo "There are no argumets!"
	echo "The command should be like ${0} [lowercase_letter] [file]"
	exit 1
fi
# I check if the input of the first argument is valid
if [[ "$1" != [a-z] ]]; then
	echo "The first argument $1 should be lowercase and a letter"
	exit 1
fi
letter="$1"
files=()
for ((args=2; args<=$#; args++));
do
	if [[ -e "${!args}" && -f "${!args}" ]]; then
		files+=( ${!args} )
	else
		echo "File ${!args} does not exist!"
done

# If the array files is empty it means that there aren't files that exist
if [[ "${#files[@]}" == 0 ]]; then
	echo "There aren't any files that exist, the program will stop!"
	exit 1
fi

# Listing the files to be edited
echo "There will be replacements done for the following files if there is anything to replace:"
for file in ${files[@]};
do
	echo "$file"
done

# Checking if you want to proceed so you can stop if something is wrong
read -p "Do you want to continue? (y/n): " answer

# Running the commands so the files will be edited
if [[ "$answer" == 'y' || "$answer" == 'Y' ]]; then
	for file in ${files[@]};
	do
		sed -i 's/[0-9]/'"$letter"'/g' $file
	done
	echo "The files have been succesfully edited!"
	exit 0
```

```
else
        echo "No changes have been made!"
        exit 0
fi
```

A2_15g.sh - Write a shell script which takes as parameters several file names. The script will display all the lines in the given files that contain only lowercase letters.

```
if [ $# -eq 0 ];
then
   echo "no args"
   exit 1
fi
for file in "$@"; do
    grep -v '[[:upper:]]' "$file"
done
```

# A3

A3_3.sh - Write a shell script which takes as parameters several file names. The script will display the ratio of the number of lowercase letters to the number of uppercase letters in each given file (ex: file1.txt l/U = 95/12).

```
#!/bin/bash

# Check if at least one file is provided
if [ "$#" -eq 0 ]; then
    echo "Usage: $0 file1 [file2 ... fileN]"
    exit 1
fi
for file in "$@"; do
    if [ ! -f "$file" ]; then
        echo "File '$file' does not exist."
        continue
    fi
    lowercase_count=$(grep -o '[a-z]' "$file" | wc -l)
     uppercase_count=$(grep -o '[A-Z]' "$file" | wc -l)   # -o -> on a new line

    # Display the ratio
    echo "$file l/U = $lowercase_count/$uppercase_count"
done
```

A3_5.sh - Write a shell script which takes as parameters several file names. For each line in the given files which contains more than 10 characters, the script will display the number of that line and its content starting from the 11th character. At the end of each file analysis, the script will display the file name and the number of lines that have been displayed before

```
    displayed_lines=0

    echo "Analysis for $file:"
    while IFS= read -r line; do
        ((total_lines++))
        if [ ${#line} -gt 10 ]; then
            ((displayed_lines++))
            ((total_displayed++))
            line_number=$(grep -n "$line" "$file" | cut -d ':' -f 1)
                                #So, if grep outputs 5:example, cut will extract 5.
            echo "Line $line_number: ${line:10}"    #display the content of the line starting from the
11th character.
        fi
    done < "$file"

    echo "Total lines displayed for $file: $displayed_lines"
   done
```

A3_6.sh - Write a shell script which takes as parameters several file names. For each given file, the script will display its name and the average number of words per line. At the end, the script will display also the average number of words per file.

```
# Function to calculate average number of words per line in a file
average_words_per_line() {
    local file="$1"
    local total_lines=$(wc -l < "$file")
    local total_words=$(wc -w < "$file")
    if [ "$total_lines" -gt 0 ]; then
        echo "Average words per line in $file: $((total_words / total_lines))"
    else
        echo "File $file is empty."
    fi
}
# Initialize variables for total words and total files
total_words=0
total_files=0
# Iterate over each file passed as parameter
for file in "$@"; do
    if [ -f "$file" ]; then
        average_words_per_line "$file"
        total_words_in_file=$(wc -w < "$file")
        total_words=$((total_words + total_words_in_file))
        total_files=$((total_files + 1))
    else
        echo "File $file not found."
    fi
done
# Calculate and display average number of words per file
if [ "$total_files" -gt 0 ]; then
    echo "Average number of words per file: $((total_words / total_files))"
else
    echo "No valid files provided."
fi
```

A3_11.sh - Write a shell script which takes as parameters several file names. The script will display the name of file which contains the highest number of words and the word count

```
#!/bin/bash

# Check if at least one file name is provided
if [ "$#" -lt 1 ]; then
    echo "too few args”
    exit 1
fi
```

```
# Initialize variables to keep track of the file with the highest word count
max_word_count=0
max_word_file=""

# Iterate over all provided file names
for file in "$@"; do
    # Check if the file exists and is readable
    if [ -f "$file" ] && [ -r "$file" ]; then
        # Count the number of words in the file
        word_count=$(wc -w < "$file")

        # Update the max_word_count and max_word_file if current file has more words
        if [ "$word_count" -gt "$max_word_count" ]; then
            max_word_count=$word_count
            max_word_file=$file
        fi
    else
        echo "File '$file' does not exist or is not readable."
    fi
done

# Display the file with the highest word count and the word count
if [ -n "$max_word_file" ]; then
    echo "The file with the highest number of words is '$max_word_file' with $max_word_count
words."
else
    echo "No valid files were provided."
fi
```

## A3_12.sh - Write a shell script which takes as parameters two words followed by several file names. The script will replace any occurrence of the first word with the second word in each line of the given files

```
if [ "$#" -lt 3 ];
then
    echo "too few args"
    exit 1
fi
word_to_replace=$1
replacement_word=$2
shift 2
for file in "$@"; do
    if [ ! -f "$file" ];
    then
        echo "File '$file' not found. Skipping."
        continue
    fi
```

```
    sed  -i "s/$word_to_replace/$replacement_word/g" "$file"
    echo "Replaced occurrences of '$word_to_replace' with ' $replacement_word' in '$file'."
done
```

## A3_14.sh - Write a shell script which takes as parameter a time interval (Mar 21 12:00-13:00). The script will display the average number of users that were connected to the server in the given time interval

```
if [ "$#" -ne 3 ]; then #nr of args not equal to 3
        echo "args not equal to 3"
        exit 1
fi
month="$1"
day="$2"
time_interval="$3"
#arg is given: 15:12-15:29-> split into 2 args, sepparating by ","
start_time=$(echo "$time_interval" | cut -d'-' -f1)
end_time=$(echo "$time_interval" | cut -d'-' -f2)
# Get login records for the specified date and time interval
logins=$(last | grep "$month $day")
logins_start_time=$(echo "$logins" | grep "$start_time")
logins_end_time=$(echo "$logins_start_time" | grep "$end_time")
usernames=$(echo "$logins_end_time" | awk '{print $1}' | sort -u)
                # sort -u(unique) to remove duplicate users
users=$(echo "$usernames" | grep -c .) #search and count the nr of lines in the var usernames
#display
echo "Usernames: "
echo "$usernames"
echo "Total no of users connected on $month $day between $time_interval: $users"
echo "    "
```

# A4

## A4_4_v2.sh - Given a list of filenames and a directory name, write a Shell script that prints for each file all the subdirectories in which it appears. Print these subdirectories sorted in decreasing order of the creation date and time.

```
#!/bin/bash

# Check if at least two arguments (one directory and at least one filename) are provided
if [ "$#" -lt 2 ]; then
    echo "too few args"
    exit 1
fi

# Extract the directory name from the first argument
directory="$1"
```

```
shift

# Check if the provided directory exists
if [ ! -d "$directory" ]; then
    echo "Directory '$directory' does not exist."
    exit 1
fi

# Iterate over each file name provided as argument
for file in "$@"; do
    echo "Searching for '$file' in directory '$directory'..."

    # "%T+ %p\n" - time and file path -r->recently
    subdirs=$(find "$directory" -type f -name "$file" -printf "%T+ %p\n" | sort -r | awk '{print $2}')

    if [ -z "$subdirs" ]; then
        echo "File '$file' not found in any subdirectory."
    else
        echo "File '$file' found in the following subdirectories:"
        for subdir in $subdirs; do
            dirname=$(dirname "$subdir")
            echo "$dirname"
        done
    fi
    echo
done
```

## A4_4.sh -

```
if [ $# -ne 2 ]; then
    echo "not 2 args"
    exit 1
fi
directory_name=$1
filename_list=$2
if [ ! -d "$directory_name" ]; then
    echo "Error: Directory '$directory_name' does not exist."
    exit 1
fi
if [ ! -f "$filename_list" ]; then
    echo "Error: File '$filename_list' does not exist."
    exit 1
fi
if [ -z "$(ls -A "$directory_name")" ]; then
    echo "Error: Directory '$directory_name' is empty."
    exit 1
```

```
fi
subdirectories=$(find "$directory_name" -mindepth 1 -type d 2>/dev/null)
while IFS= read -r subdirectory; do
    echo "Searching in directory: $subdirectory"
    if [ ! -r "$subdirectory" ]; then
        echo "Warning: No read permission for directory '$subdirectory'"
        continue
    fi
    while IFS= read -r filename; do
        if [ -e "$subdirectory/$filename" ]; then
            echo "File '$filename' found in subdirectory '$subdirectory'"
        fi
    done < "$filename_list"
done <<< "$subdirectories"
```

A4_11.sh - Given a list of files and a directory, print for each filename all the subdirectories in which it appears, ordered decreasingly by the file size (a file can have the same name but different sizes in different subdirectories).

```
# Check if at least two arguments (one directory and at least one filename) are provided
if [ "$#" -lt 2 ]; then
    echo "Usage: $0 directory_name file1 [file2 ... fileN]"
    exit 1
fi

# Extract the directory name from the first argument
directory="$1"
shift

# Check if the provided directory exists
if [ ! -d "$directory" ]; then
    echo "Directory '$directory' does not exist."
    exit 1
fi

# Iterate over each file name provided as argument
for file in "$@"; do
    echo "Searching for '$file' in directory '$directory'..."

# The -exec ls -lh {} + command lists the file details including the file size in human-readable format.{print $9, $5}' command extracts the file paths and sizes

subdirs=$(find "$directory" -type f -name "$file" -exec ls -lh {} + | sort -k 5 -r | awk '{print $9, $5}')

    if [ -z "$subdirs" ]; then
        echo "File '$file' not found in any subdirectory."
    else
        echo "File '$file' found in the following subdirectories (sorted by size):"
        echo "$subdirs"
```

```
    fi
    echo
done
```

A4_12_v2.sh - Write a Shell program that, given a directory (as a parameter), creates a list of all the names that appear in it and its subdirectories (files, directories), and for each file it prints the maximum number of repeating lines (in the same file).

```
#!/bin/bash

# Check if a directory is provided as a parameter
if [ "$#" -ne 1 ]; then
    echo "Usage: $0 directory"
    exit 1
fi

# Check if the provided parameter is a valid directory
if [ ! -d "$1" ]; then
    echo "$1 is not a directory"
    exit 1
fi

# Function to find the maximum number of repeating lines in a file
max_repeating_lines() {
    file=$1
    if [ -f "$file" ]; then
        awk '{ count[$0]++ } END { max = 0; for (line in count) { if (count[line] > max) { max =
count[line] } } print max }' "$file"
    fi
}

# List all names in the directory and its subdirectories
find "$1" -print | while read -r entry; do
    echo "Name: $entry"
    if [ -f "$entry" ]; then
        max_repeats=$(max_repeating_lines "$entry")
        echo "Maximum repeating lines in $entry: $max_repeats"
    fi
done
```

A4_16.sh - Write a Shell program that receives as parameters two directory names and will copy the branch specified by the second directory to be a sub branch in the first directory, but it will copy only .txt files.

```
#!/bin/bash

# Check if two directory names are provided
if [ "$#" -ne 2 ]; then
```

```
    echo "not 2 args"
    exit 1
fi

source_dir="$1"
target_dir="$2"

if [ ! -d "$source_dir" ]; then
    echo "Source directory '$source_dir' does not exist."
    exit 1
fi

if [ ! -d "$target_dir" ]; then
    echo "Target directory '$target_dir' does not exist."
    exit 1
fi

# Copy .txt files from source directory to target directory
find "$source_dir" -type f -name "*.txt" -exec cp -t "$target_dir" {
```

# A5

## A5_1_v2.c - Write a C Program that receives a command line argument a filename and prints to standard output the content of this text file, printing also an empty line after each line of text. Make no assumption regarding the maximum length of a line.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    // Check if a filename is provided
    if (argc != 2) {
        fprintf(“not 2 args”);
        return 1;
    }

    // Open the file
    FILE *file = fopen(argv[1], "r"); // r- read permission
    if (file == NULL) {
        perror("Error opening file");
        return 1;
    }

    // Read and print the file line by line
    char *line = NULL;
    size_t len = 0;
    ssize_t read;
    while ((read = getline(&line, &len, file)) != -1) {
        printf("%s\n", line);  // Print the line followed by an empty line
    }

    // Free allocated memory and close the file
    free(line);
    fclose(file);

    return 0;
}
```

## A5_5.c - Write a C Program that receives as command line arguments two words and the name of a file. The program will print at stdout the content of the file, replacing the occurrences of the first word with the second

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]) {
    // Check if correct number of command line arguments are provided
    if (argc != 4) {
        printf("Usage: %s <word_to_replace> <replacement_word> <filename>\n", argv[0]);
        return 1;
```

```
    }

    // Open the file for reading
    FILE *file = fopen(argv[3], "r");
    if (file == NULL) {
        printf("Error: Unable to open file %s\n", argv[3]);
        return 1;
    }

    // Read the file line by line and replace occurrences of the first word with the second
    char line[1000];
    while (fgets(line, sizeof(line), file) != NULL) {
        // Replace occurrences of the first word with the second
        char *word = strtok(line, " \t\n");
        while (word != NULL) {
            if (strcmp(word, argv[1]) == 0) {
                printf("%s ", argv[2]); // Print the replacement word
            } else {
                printf("%s ", word); // Print the original word
            }
            word = strtok(NULL, " \t\n");
        }
        printf("\n");
    }

    // Close the file
    fclose(file);

    return 0;
}
```

A5_10.c - Write a C Program that receives as command line argument a name of a file. The program prints at stdout the content of this file, erasing all empty lines from the file. Make no assumptions regarding the maximum length of a line.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(“not 2 args”);
        return 1;
    }

    FILE *file = fopen(argv[1], "r");
    if (file == NULL) {
        perror("Error opening file");
        return 1;
    }
```

```
    char *line = NULL;
    size_t len = 0;
    ssize_t read;       //   ‘ ‘ -space     \t -tab      \n-newline

    while ((read = getline(&line, &len, file)) != -1) { // cat timp se mai poate citi
        int isEmpty = 1;   // Check if the line is not empty (ignoring whitespace)
        for (ssize_t i = 0; i < read; i++) {
            if (line[i] != ' ' && line[i] != '\t' && line[i] != '\n') {
                isEmpty = 0;
                break;
            }
        }
        if (!isEmpty) {     // Print the line if it's not empty
            printf("%s", line);
        }
    }

    free(line);
    fclose(file);

    return 0;
}
```

A5_13.c - Write a C Program that receives as command line arguments a name of a file and several words. The program will print at stdout the content of the file, deleting the occurrences of all words provided as arguments

```
#include <stdio.h>
        #include <stdlib.h>
        #include <string.h> // Function to check if a word is in the list of words to delete

        int is_word_to_delete(char *word, char *words[], int word_count) {
            for (int i = 0; i < word_count; i++) {
                if (strcmp(word, words[i]) == 0) { //compare the provided words with others
                    return 1;
                }
            }
            return 0;
        }

        // Function to print the contents of the file with specified words removed
        void print_file_without_words(FILE *file, char *words[], int word_count) {
            char *line = NULL;
            size_t len = 0;
            ssize_t read;

            while ((read = getline(&line, &len, file)) != -1) {
                char *token = strtok(line, " \t\n"); //split the line into words
```

```
            int first = 1;

            while (token != NULL) {
                if (!is_word_to_delete(token, words, word_count)) {
                    if (!first) {
                        printf(" ");
                    }
                    printf("%s", token);
                    first = 0;
                }
                token = strtok(NULL, " \t\n");
            }
            printf("\n");
        }
        free(line);
    }

    int main(int argc, char *argv[]) {
        if (argc < 3) {
            fprintf(“not 3 args”);
            return 1;
        }

        char *filename = argv[1];
        char **words = &argv[2];
        int word_count = argc - 2;

        FILE *file = fopen(filename, "r");
        if (file == NULL) {
            perror("Error opening file");
            return 1;
        }

        print_file_without_words(file, words, word_count);

        fclose(file);
        return 0;
    }
```

A5_15_v2.c - Write a C Program that counts the number of letters on each line of a text file. Make no assumptions regarding the maximum length of a line.

```
int main(int args) {
    FILE* ptr;
    char ch;
    int count = 0;
```

```
    ptr = fopen("latin.txt", "r");
    if (ptr == NULL) {
        printf("file cannot be opened.\n");
        return 1;
    }
    printf("letter count on each line from file:\n");
    while ((ch = fgetc(ptr))) {
        if (ch != '\n') {
            if (ch != ' ') {
                count++;
            }
        } else {
            printf("%d\n", count);
            count = 0;
        }
    }
    fclose(ptr);
    return 0;
}
```

**METHOD 2 :**

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

// Function to count the number of letters in a line
int count_letters(const char *line) {
    int count = 0;
    for (int i = 0; line[i] != '\0'; i++) {
        if (isalpha((unsigned char)line[i])) { // isalpha- if letter
            count++;
        }
    }
    return count;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(“not 2 args”);
        return 1;
    }

    char *filename = argv[1];
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        perror("Error opening file");
        return 1;
    }

    char *line = NULL;
    size_t len = 0;
```

```
    ssize_t read;
    int line_number = 1;

    while ((read = getline(&line, &len, file)) != -1) {
        int letter_count = count_letters(line);
        printf("Line %d: %d letters\n", line_number, letter_count);
        line_number++;
    }

    free(line);
    fclose(file);

    return 0;
}
```

# A6

A6_3.c – We have a file that contains N integer numbers. Using two types of processes (one for determining the minimum and the other to determine the maximum value from an array), write a program that determines the kth element in increasing order of the integer array, without sorting the array

```
 #include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int find_min(int *arr, int size) {    // Function to find the minimum value in the array
    int min = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] < min) {
            min = arr[i];
        }
    }
    return min;
}

int find_max(int *arr, int size) {  // Function to find the maximum value in the array
    int max = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}

int partition(int *arr, int left, int right, int pivot) { // Function to partition the array and return the position of the pivot
    int pivotValue = arr[pivot];
    int storeIndex = left;
    int temp = arr[pivot];
    arr[pivot] = arr[right];
    arr[right] = temp;
```

```
    for (int i = left; i < right; i++) {
        if (arr[i] < pivotValue) {
            temp = arr[i];
            arr[i] = arr[storeIndex];
            arr[storeIndex] = temp;
            storeIndex++;
        }
    }

    temp = arr[storeIndex];
    arr[storeIndex] = arr[right];
    arr[right] = temp;

    return storeIndex;
}

int quickselect(int *arr, int left, int right, int k) { // Function to find the k-th smallest element in the array
    if (left == right) {
        return arr[left];
    }

    int pivotIndex = left + (right - left) / 2;
    pivotIndex = partition(arr, left, right, pivotIndex);

    if (k == pivotIndex) {
        return arr[k];
    } else if (k < pivotIndex) {
        return quickselect(arr, left, pivotIndex - 1, k);
    } else {
         return quickselect(arr, pivotIndex + 1, right, k);
    }
}

int main(int argc, char *argv[]) {
    if (argc < 3) {
        fprintf(“not 3 args”);
        return 1;
    }

    int k = atoi(argv[1]);
    int n = argc - 2;
    int *arr = malloc(n * sizeof(int));
    if (arr == NULL) {
        perror("malloc");
        return 1;
    }

    for (int i = 0; i < n; i++) {
        arr[i] = atoi(argv[i + 2]);
    }

    // Create two child processes
    pid_t pid_min = fork();
    if (pid_min == 0) {
        // Child process to find the minimum
        int min = find_min(arr, n);
        printf("Minimum value: %d\n", min);
        exit(0);
```

```
    }

    pid_t pid_max = fork();
    if (pid_max == 0) {
        // Child process to find the maximum
        int max = find_max(arr, n);
        printf("Maximum value: %d\n", max);
        exit(0);
    }

    // Wait for both child processes to complete
    wait(NULL);
    wait(NULL);

    // Find the k-th smallest element
    if (k > 0 && k <= n) {
        int kth_smallest = quickselect(arr, 0, n - 1, k - 1);
        printf("The %d-th smallest element is %d\n", k, kth_smallest);
    } else {
        fprintf(stderr, "Invalid value of k: %d. It should be between 1 and %d.\n", k, n);
    }

    free(arr);
    return 0;
}
```

## A6_9.c - An array of numbers is given x1, x2, ..., xn. Compute the partial sums x1, x1+x2, x1+x2+x3, ...,

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

// Function to compute partial sum
int compute_partial_sum(int *arr, int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += arr[i];
    }
    return sum;
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        Fprintf(“not 2 args”);
        return 1;
    }

    int n = argc - 1;
    int *arr = malloc(n * sizeof(int));
    if (arr == NULL) {
        perror("malloc");
        return 1;
    }
```

```
    for (int i = 0; i < n; i++) {
        arr[i] = atoi(argv[i + 1]);
    }

    // Fork a child process for each partial sum x1+x2  ,  x1+x2+....+xn
    for (int i = 1; i <= n; i++) {
        pid_t pid = fork();
        if (pid == -1) {
            perror("fork");
            free(arr);
            return 1;
        } else if (pid == 0) { // Child process
            int partial_sum = compute_partial_sum(arr, i);
            printf("Partial sum up to x%d: %d\n", i, partial_sum);
            free(arr);
            exit(0);
        }
    }

    // Wait for all child processes to complete
    for (int i = 0; i < n; i++) {
        wait(NULL);
    }

    free(arr);
    return 0;
}
```

A6_10.c - An array of letters is given. Using a separate process for each vowel, delete all vowels from the array.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

void remove_vowel(char *arr, char vowel) {
    int j = 0;
    int len = strlen(arr);
    for (int i = 0; i < len; i++) {
            if (arr[i] != vowel && arr[i] != vowel - 32) {
            arr[j++] = arr[i];
        }
    }
    arr[j] = '\0'; // Null-terminate the new array
    printf("%s\n", arr); // Print the result after removing the vowel
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
```

```
        fprintf(stderr, "Usage: %s string_of_letters\n", argv[0]);
        return 1;
    }

    char *input = argv[1];
    char vowels[] = "aeiou";
    int num_vowels = strlen(vowels);
    char *arr = malloc((strlen(input) + 1) * sizeof(char));

    if (arr == NULL) {
        perror("malloc");
        return 1;
    }

    strcpy(arr, input);

    // Fork a child process for each vowel
    for (int i = 0; i < num_vowels; i++) {
        pid_t pid = fork();
        if (pid == -1) {
            perror("fork");
            free(arr);
            return 1;
        } else if (pid == 0) { // Child process
            remove_vowel(arr, vowels[i]);
            exit(0);
        }
    }

    // Wait for all child processes to complete
    for (int i = 0; i < num_vowels; i++) {
        wait(NULL);
    }

    free(arr); // Free memory in the parent process
    return 0;
}
```

A6_12.c - Compute the sum of an array of numbers using ***divide et impera*** method: a process splits the array in two sub-arrays and gives them to two other child processes to compute their sums, then adds the results obtained. The child processes use the same technique (split, etc. …).

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

// Function to compute the sum of an array segment
int compute_sum(int *arr, int start, int end) {
    int sum = 0;
    for (int i = start; i <= end; i++) {
        sum += arr[i];
    }
```

```
    return sum;
}

// Recursive function to compute the sum using divide and conquer
int divide_and_conquer_sum(int *arr, int start, int end) {
    if (start == end) {
        return arr[start];
    }

    int mid = (start + end) / 2;
    int fd[2]; // file descriptor for pipe
    pid_t pid1, pid2;

    // Create a pipe
    if (pipe(fd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    // Fork the first child process
    if ((pid1 = fork()) == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (pid1 == 0) { // First child process
        close(fd[0]); // Close unused read end
        int left_sum = divide_and_conquer_sum(arr, start, mid);
        write(fd[1], &left_sum, sizeof(left_sum));
        close(fd[1]); // Close write end
        exit(0);
    }

    // Fork the second child process
    if ((pid2 = fork()) == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (pid2 == 0) { // Second child process
        close(fd[0]); // Close unused read end
        int right_sum = divide_and_conquer_sum(arr, mid + 1, end);
        write(fd[1], &right_sum, sizeof(right_sum));
        close(fd[1]); // Close write end
        exit(0);
    }

    // Parent process
    close(fd[1]); // Close unused write end
    int left_sum, right_sum;
    read(fd[0], &left_sum, sizeof(left_sum));
    read(fd[0], &right_sum, sizeof(right_sum));
    close(fd[0]); // Close read end

    // Wait for both child processes to finish
    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);

    return left_sum + right_sum;
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s num1 num2 num3 ...\n", argv[0]);
        return EXIT_FAILURE;
    }

    int n = argc - 1;
    int *arr = malloc(n * sizeof(int));
    if (!arr) {
```

```
        perror("malloc");
        return EXIT_FAILURE;
    }

    for (int i = 0; i < n; i++) {
        arr[i] = atoi(argv[i + 1]);
    }

    int total_sum = divide_and_conquer_sum(arr, 0, n - 1);
    printf("Total sum: %d\n", total_sum);

    free(arr);
    return EXIT_SUCCESS;
}
```

A6_13.c - Calculate the product of an array n of numbers using ***divide et impera*** method: a process splits the array in two sub-arrays and gives them to two other child processes to compute their product, then multiplies the results obtained from the two. The two child processes apply the same technique (split, etc…)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

// Function to compute the product of an array segment
int compute_sum(int *arr, int start, int end) {
    int product = 1;;
    for (int i = start; i <= end; i++) {
        product *= arr[i];
    }
    return product;
}

// Recursive function to compute the product using divide and conquer
int divide_and_conquer_product(int *arr, int start, int end) {
    if (start == end) {
        return arr[start];
    }

    int mid = (start + end) / 2;
    int fd[2]; // file descriptor for pipe
    pid_t pid1, pid2;

    // Create a pipe
    if (pipe(fd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    // Fork the first child process
    if ((pid1 = fork()) == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (pid1 == 0) {        // First child process
        close(fd[0]);                // Close unused read end
        int left_product = divide_and_conquer_product(arr, start, mid);
        write(fd[1], &left_product, sizeof(left_product));
        close(fd[1]);                // Close write end
        exit(0);
    }

    // Fork the second child process
```

```
    if ((pid2 = fork()) == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (pid2 == 0) {               // Second child process
        close(fd[0]);                     // Close unused read end
        int right_product = divide_and_conquer_product(arr, mid + 1, end);
        write(fd[1], &right_product, sizeof(right_product));
        close(fd[1]);                     // Close write end
        exit(0);
    }

    // Parent process
    close(fd[1]);                   // Close unused write end
    int left_product, right_product;
    read(fd[0], &left_product, sizeof(left_product));
    read(fd[0], &right_product, sizeof(right_product));
    close(fd[0]);                   // Close read end

    // Wait for both child processes to finish
    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);

    return left_product * right_product;
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        Fprint(f"not 2 args");
        return EXIT_FAILURE;
    }

    int n = argc - 1;
    int *arr = malloc(n * sizeof(int));
    if (!arr) {
        perror("malloc");
        return EXIT_FAILURE;
    }

    for (int i = 0; i < n; i++) {
        arr[i] = atoi(argv[i + 1]);
    }

    int total_product = divide_and_conquer_product(arr, 0, n - 1);
    printf("Total product: %d\n", total_sum);

    free(arr);
    return EXIT_SUCCESS;
}
```

# OTHER

1. Write a bash script that counts all the C files from a given directory and all of its subdirectories.

```
#!/bin/bash
if [ $# -lt 1 ]; then
echo "Insufficient arguments"
exit 1
fi
find $1 -type f | grep -E -c "\.c$"
```

1a. Write a bash script that counts all the lines of code in the C files from the directory given as command-line argument, excluding lines that are empty or contain only spaces.

```
#!/bin/bash
if [ -z "$1" ]; then
echo "No parameters given"
exit 1
fi
if [ ! -d "$1" ]; then
echo "Parameter is not a folder"
exit 1
fi
total=0
for f in $(ls "$1" | grep -E "\.c$"); do
if test -f "$1/$f"; then
nr_lines=$(grep -E -c -v "^[ \t]*$" "$1/$f")
echo "$f: $nr_lines"
total=$((total+nr_lines))
fi
done
echo "Total lines: $total"
```

1b. Write a bash script that counts all the lines of code in the C files from the directory given as command-line argument and all its subdirectories, excluding lines that are empty or contain only spaces.

```
#!/bin/bash
if [ -z "$1" ]; then
echo "No parameters given"
exit 1
fi
if [ ! -d "$1" ]; then
echo "Parameter is not a folder"
exit 1
fi
total=0
for f in $(find "$1" -type f | grep -E "\.c$"); do
nr_lines=$(grep -E -c -v "^[ \t]*$" $f)
echo "$f - $nr_lines"
total=$((total+nr_lines))
done
echo "Total lines: $total"
```

2. Write a bash script that receives any number of command line arguments and prints on the screen, for each argument, if it is a file, a directory, a number or something else.

```
#!/bin/bash
while [ ! $# -eq 0 ]; do
arg=$1
if test -f $arg; then
echo "$arg is a regular file"
elif [ -d $arg ]; then
echo "$arg is a directory"
elif echo $arg | grep -E -q "^[0-9]+$"; then
# the regular expression here can be adapted to match any type of number, as needed
echo "$arg is an integer number"
else
echo "$arg is something else"
fi
shift
Done
```

3. Write a bash script that keeps reading strings from the keyboard until the name of a readable regular file is given.

```
#!/bin/bash
fname=""
while [ ! -f "$fname" ]; do
read -p "Enter a string: " fname
done
```

4. Write a bash script that sorts the file given as command line arguments in ascending order according to their file size in bytes.

```
#!/bin/bash

# Check if no file is provided
if [ "$#" -eq 0 ]; then
    echo "No files provided."
    exit 1
fi

# Iterate over each file provided as argument
for file in "$@"; do
    # Check if the file exists
    if [ ! -e "$file" ]; then
        echo "File '$file' does not exist."
        continue
    fi

    # Get file size in bytes
    file_size=$(stat -c "%s" "$file")

    # Print file size and filename
    echo "$file_size $file"
done | sort -n | awk '{print $2}'
```

5. Write a script that receives as command line arguments pairs consisting of a filename and a word. For each pair, check if the given word appears at least 3 times in the file and print a corresponding message.

```
#!/bin/bash
if [ $# -lt 2 ]; then
echo "Please provide at least 2 arguments"
exit 1
fi
if [ $(($# % 2)) -eq 1 ]; then
echo "You must provide an even number of arguments"
exit 1
fi
while [ $# -gt 1 ]; do
file=$1
word=$2
if [ ! -f "$file" ]; then
echo "Name $file is not a file"
else count=$(grep -E -o "\<$word\>" "$file" | wc -l)
if [ $count -ge 3 ]; then
echo "Word $word appears $count times in file $file"
fi
fi
shift 2
done
if [ $# -eq 1 ]; then
echo "Warning: final pair is incomplete"
fi
```

6. Find recursively in a given directory all the symbolic links, and report those that point to files/directories that no longer exist. Use option -L to test if a path is a symbolic link, and option -e to test if it exists (will return false if the target to which the link points does not exist)

```
#!/bin/bash
for link in $(find "$1" -type l); do
if [ ! -e "$link" ]; then
echo "Link $link is not valid"
fi
Done
```

7. Write a bash script that receives a folder name as argument. Find recursively in the folder the number of times each file name is repeated.

```
#!/bin/bash
if [ -z "$1" ]; then
echo "Please provide one argument"
exit 1
fi
if [ ! -d "$1" ]; then
echo "Argument must be a directory"
exit 1
fi
find "$1" -type f | awk -F/ '{print $NF}' | sort | uniq -c
```

8. Write a script that receives program/process names as command line arguments. The script will monitor all the processes in the system, and whenever a program with one of those names is run, the script will kill it and display a message. (see commands ps, kill, killall). Alternativ, comenzile pkill/pgrep pot fi folosite.

```
#!/bin/bash
if [ $# -eq 0 ]; then
echo "Provide at least one name"
exit 1
fi
while true; do
for process in $@; do
PIDs=""
PIDs=$(ps -ef | awk '{print $8" "$2}' | grep -E "\<$process " | awk '{print $2}')
if [ -n "$PIDs" ]; then
kill -9 $PIDs
fi
done
sleep 3
Done
```

9. Consider a file containing a username on each line. Generate a comma-separated string with email addresses of the users that exist. The email address will be obtained by appending "@scs.ubbcluj.ro" at the end of each username. Make sure the generated string does NOT end in a comma.

```
#!/bin/bash
if [ -z "$1" ]; then
echo "Please provide one input file"
exit 1
fi
if [ ! -f "$1" ]; then
echo "The given argument is not a file"
exit 1
fi
result=""
for u in $(cat "$1"); do
result="$u@scs.ubbcluj.ro,$result"
doneresult=$(echo $result | sed -E "s/,$//")
echo $result
```

1. Write a script that reads filenames and check for each file how many words contains on the first line and the size of the file. Perform all required validations on the input data.

```
#!/bin/bash

# Function to check if a file is readable
is_readable_file() {
    if [ ! -f "$1" ]; then
        echo "File '$1' does not exist."
        return 1
    elif [ ! -r "$1" ]; then
        echo "File '$1' is not readable."
        return 1
    fi
```

```
    return 0
}

# Read filenames from user input
read -p "Enter filenames separated by spaces: " -a filenames

# Iterate over each filename provided
for file in "${filenames[@]}"; do
    # Validate file
    if is_readable_file "$file"; then
        # Count words in the first line
        first_line=$(head -n 1 "$file")
        word_count=$(echo "$first_line" | wc -w)

        # Get the size of the file
             file_size=$(stat -c%s "$file")

        # Display the results
        echo "File: $file"
        echo "Number of words in the first line: $word_count"
        echo "Size of the file: $file_size bytes"
        echo
    fi
done
```

Write a shell script that receives as argument a natural number N and generate N text files:
- the name of the files will be of the form: file_X, where X={1,2,…, N}
- each generated file will contain online lines between X and X+5 of the file /etc/passwd

```
#!/bin/bash

# Check if the correct number of arguments is provided
if [ "$#" -ne 1 ]; then
    echo "Usage: $0 N"
    exit 1
fi

# Validate that the argument is a natural number
if ! [[ "$1" =~ ^[1-9][0-9]*$ ]]; then
    echo "Error: Argument must be a natural number."
    exit 1
fi

# Get the value of N
N="$1"

# Generate N files
for ((X=1; X<=N; X++)); do
    file_name="file_$X"
    start_line=$X
    end_line=$((X + 5))

    # Extract lines from /etc/passwd and save to the new file
    sed -n "${start_line},${end_line}p" /etc/passwd > "$file_name"
done

echo "$N files generated successfully."
```

Calculate the average of all process ids in the system per user.

```
#!/bin/bash

# Get the list of all processes with their users and PIDs
processes=$(ps -e -o user=,pid=)

# Check if the processes variable is empty
if [ -z "$processes" ]; then
    echo "No processes found."
    exit 1
fi

# Initialize associative arrays for storing sums and counts
declare -A pid_sums
declare -A pid_counts
```

```
# Process each line of the output
while read -r user pid; do
    # Add PID to the sum for this user
    pid_sums["$user"]=$((pid_sums["$user"] + pid))
    # Increment the count for this user
    pid_counts["$user"]=$((pid_counts["$user"] + 1))
done <<< "$processes"

# Calculate and print the average PID for each user
echo "Average PID per user:"
for user in "${!pid_sums[@]}"; do
    sum=${pid_sums["$user"]}
    count=${pid_counts["$user"]}
    avg=$(echo "scale=2; $sum / $count" | bc)
    echo "User: $user, Average PID: $avg"
done
```

Write a shell script that for each command line parameter will do:
- if it's a file, print the name, number of characters and lines in the file
- if it's a directory, print the name and how many files it contains (including subdirectories files)

#!/bin/bash

```
# Check if any arguments are provided
if [ "$#" -eq 0 ]; then
    echo "Please provide at least one file or directory as an argument."
    exit 1
fi

# Iterate over each command-line parameter
for item in "$@"; do
    if [ -f "$item" ]; then
        # If it's a file, print the name, number of characters, and number of lines
        num_chars=$(wc -m < "$item")
        num_lines=$(wc -l < "$item")
        echo "File: $item"
        echo "Number of characters: $num_chars"
        echo "Number of lines: $num_lines"
        echo
    elif [ -d "$item" ]; then
        # If it's a directory, print the name and number of files (including subdirectory files)
        num_files=$(find "$item" -type f | wc -l)
        echo "Directory: $item"
        echo "Number of files (including subdirectory files): $num_files"
        echo
    else
        # If it's neither a file nor a directory, print an error message
        echo "$item is neither a file nor a directory."
        echo
    fi
done
```

Write a shell script that received triplets of command line arguments consisting a filename, a word and a number (validate input data). For each such triplet, print all the lines in the filename that contain the word exactly k times.

```
#!/bin/bash

# Function to validate input data
validate_triplet() {
    if [ ! -f "$1" ]; then
        echo "Error: $1 is not a valid file."
        return 1
    fi

    if [[ ! "$3" =~ ^[0-9]+$ ]]; then
        echo "Error: $3 is not a valid number."
        return 1
    fi
```

```
    return 0
}

# Function to print lines containing the word exactly k times
print_lines_with_word_k_times() {
    local file="$1"
    local word="$2"
    local k="$3"

    grep -E -o "\b$word\b" "$file" | grep -c -o "\b$word\b" | awk -v k="$k" 'NR == k'
}

# Check if the number of arguments is a multiple of 3
if [ $(( $# % 3 )) -ne 0 ]; then
    echo "Usage: $0 filename1 word1 number1 [filename2 word2 number2 ...]"
    exit 1
fi

# Iterate over each triplet of arguments
while [ "$#" -gt 0 ]; do
    filename="$1"
    word="$2"
    number="$3"

    # Validate the triplet
    if validate_triplet "$filename" "$word" "$number"; then
        echo "Lines in $filename containing the word '$word' exactly $number times:"
        # Use grep and awk to find and print lines that contain the word exactly k times
        grep -E -o "\b$word\b" "$filename" | awk -v word="$word" -v k="$number" '
        {
            count = gsub(word, "&")
            if (count == k) {
                print NR ": " $0
            }
        }'
    fi

    # Shift to the next triplet
    shift 3
done
```

```
#!/bin/bash

# Function to find and list files with write permissions for everybody
find_files_with_write_permissions() {
    find . -type f -perm -o=w
}
```

Write a script that finds recursively in the current folder and displays all the regular files that have write permissions for everybody (owner, group, other). Then the script removes the write permissions from everybody. Hint: use chmod's symbolic permissions mode (see the manual).

```
# Function to remove write permissions for everybody
remove_write_permissions() {
    chmod a-w "$1"
}

# Find files with write permissions for everybody
files_with_write_permissions=$(find_files_with_write_permissions)

# Check if any files are found
if [ -z "$files_with_write_permissions" ]; then
    echo "No files with write permissions for everybody were found."
    exit 0
fi

# Display the found files
echo "Files with write permissions for everybody:"
echo "$files_with_write_permissions"

# Remove write permissions for each found file
echo "Removing write permissions for everybody..."
```

```
while IFS= read -r file; do
    remove_write_permissions "$file"
    echo "Removed write permissions for $file"
done <<< "$files_with_write_permissions"

echo "All permissions updated."
```

Write a shell script that receives any number of words as command line arguments, and continuously reads from the keyboard one file name at a time. The program ends when all words received as parameters have been found at least once across the given files.

```
#!/bin/bash

# Check if at least one word is provided
if [ "$#" -lt 1 ]; then
    echo "Usage: $0 word1 word2 ... wordN"
    exit 1
fi

# Array to keep track of words to be found
declare -A words_to_find

# Initialize the words_to_find array
for word in "$@"; do
    words_to_find["$word"]=0
done

# Function to check if all words have been found
all_words_found() {
    for word in "${!words_to_find[@]}"; do
        if [ "${words_to_find[$word]}" -eq 0 ]; then
            return 1
        fi
    done
    return 0
}

# Read file names from the keyboard
while true; do
    read -p "Enter a file name: " filename

    # Check if the file exists and is readable
    if [ ! -f "$filename" ] || [ ! -r "$filename" ]; then
        echo "File '$filename' does not exist or is not readable."
        continue
    fi

    # Check the file for each word
    for word in "${!words_to_find[@]}"; do
        if grep -q "$word" "$filename"; then
            words_to_find["$word"]=1
        fi
    done

    # Check if all words have been found
    if all_words_found; then
        echo "All words have been found in the given files."
        exit 0
    fi
done
```